Computation Offloading: Is it Practical and Feasible?

Abdul Haseeb Shujja, Imran Saleem, Sher Afghan

***Abstract* – Mobile phones are usually poor in terms of battery, computation power and network bandwidth, which result in applications with limited functionality in terms of complex computations. A solution to this problem is "Computation Offloading". By sending resource intensive computations to a server, precious resources like battery and processing power can be saved on a mobile device. In the past few years, many techniques have been proposed to approach this matter ranging from utilizing virtual machines with cloud servers and mobile network infrastructure to using nearby mobile devices to perform computation intensive tasks. This paper is a survey on existing techniques and systems for computation offloading and in light of those analyzes whether computation offloading is feasible to be deployed commercially with the current infrastructure and technology available. It also analyzes the major problems and identifies possible future research areas for computation offloading which may help in overcoming the current issues.**

***Index Terms* – Computing, Offloading, Ubiquitous Computing**

I. INTRODUCTION

It is quite fair to say that this is the era of voracious mobile computing. The greatest obstacle in today's mobile computing is the limited resources of mobile devices. We have high-speed processors, GPS, high-resolution screens and much more in our mobile devices but we need power to keep these things alive. Short lifespan of batteries is the greatest obstacle in meeting our mobile computing requirements. We need to reduce the gap between the required and available power. We either increase the lifespan of the batteries or somehow reduce the computation on our mobile devices. In near future, we do not see a significant increase in battery lifespan [1]. Mobile devices are resource constrained specially in terms of energy. A lot of work has been done to overcome the bounds of limited mobile devices resources by means of reducing computations on mobile devices and one of them is by using the art of 'code offloading' which shares the same idea behind RPCs where procedural calls are made to a remote resource intensive server and this way performing computations locally on smart phone can be avoided. Connectivity between mobile client and the remote server plays a pivotal role in code offloading because of the battery-backed power of mobile devices and inherently unreliable wireless channel of communication. Code offloading in mobile devices can yield fascinating results in lab environment.

In this paper, we analyze some of the major approaches for offloading and identify areas of research that we need to focus in the future to make offloading commercially viable. We analyzed different architectures for code off-loading to filter down the barriers of this technique of overcoming the resource constraints of mobile devices available today to meet our rapidly growing needs of mobile computing.

As mentioned above, energy conservation is the primary concern when it comes to utilizing the full potential of smart phones. Different studies have shown that longer battery life is the top concern for smart phone users [2, 3]. Computation offloading can help reduce battery consumption, but there needs to be some decision making involved as to when this is feasible or not. Other than computations, the second most energy-consuming task is communication in a mobile environment, certain rules need to be enforced when offloading which would determine whether offloading is beneficial or not in a current scenario. If energy required to offload data is less than what is required to compute it locally then the better choice is to offload the data. Previous studies have shown that applications with large computation-to-communication ratio benefit more from code offloading [4].

Other than energy there are several other factors that affect this decision-making process. Available bandwidth, its cost and latency are a few major ones. Many algorithms have been proposed to make these decisions possible to increase performance or conserve energy. These techniques make these decisions based on several factors which include bandwidths, server speeds, available memory, server loads, and the amounts of data exchanged between server and mobile systems. The solutions include partitioning programs [5, 6, 7, 8] and predicting parametric variations in application behavior and execution environment [9, 10].

Once these offloading decisions are made, the next question that arises is what to offload? Obviously, parts of code, which involve interactions with the user or environment, cannot be offloaded to the server. However, other parts of the code, which involve intense computations, can be offloaded to the server. The algorithm used for code offloading also needs to specify whether to offload all this code to the server or just send parts of it. Although each algorithm describes its own way of partitioning code, but, these can be grouped into two major categories i.e. [4]

- Static Partitioning
- Dynamic Partitioning

In case of Static Partitioning, the programmer must specify which part of the code should be offloaded to run at the server. In addition, at run-time if the offloading parameters allow it, that part of the code is offloaded to the server.

While in Dynamic Partitioning of code, the programmer might also have to specify off loadable code, the final decision on whether the code should be offloaded or not is dependent on the algorithm used. This decision could be

Abdul Haseeb Shujja, Imran Saleem and Sher Afghan, Department of Computer Science, School of Professional Advancement, University of Management and Technology, Email: abdul.shujja@umt.edu.pk, imran.saleem@umt.edu.pk, sher.afghan@umt.edu.pk. Manuscript received Dec 08, 2016; revised on June 12, 2017; accepted on Aug 31, 2017.

 ISSN – 1999-4974

based on network conditions, previous behavior of the code or the amount of data to be transferred.

A significant amount of research has been done over the past 15-20 years concerning computation offloading about making it feasible, reliable decision-making and developing proper infrastructures. In the latter half of 1990's the focus was more on making code offloading more feasible for mobile environments [11,12,13,14,15,16,17,18,19] as the mobile bandwidths at that time were insufficient for any practical implementation of offloading. in the early 2000's the focus shifted towards the decision-making process [5,6,8,9,20,21,22,23,24,25] involved and then later on with advances in cloud computing, increased network bandwidths and virtualization technology new doors were opened with regards to the infrastructure of code offloading environments [7,26,27,28,29,30,31]. These technologies have made computation offloading more practical in a real-world environment than just in lab setups. In the recent years, a lot of solutions have been proposed which can be used to implement offloading on a commercial scale [32, 33, 34, 35, 36].These algorithms use either cloud environments or virtualization technologies or a combination of both in most cases.

The purpose of this paper is to analyze some of the leading techniques covering computation offloading while asking the question; is it feasible for the current mobile networks we have and if not what needs to be done to make it more practical? For the purpose, we have done a detailed analysis of some of these techniques. The paper is organized as follows: Section 2 explains the current state of the research, which has been done concerning code offloading. Section 3 discusses the major enabling technologies in this field. In section 4 we discuss the problems we have identified as being the major reasons behind code offloading not being practical yet. Finally, section 5 concludes the paper with some future research topics, which could help, eliminate the problems identified.

II. CURRENT STATE OF CODE OFFLOADING

As mentioned in the introduction there has been a lot of research on the subject of code offloading over the years. In this section, we describe a summary of some of the latest studies in this subject. However, before that it is necessary to understand the basic reasons why computation offloading is so important and what are the factors affecting our decision-making process when offloading code to a server. Section 2.1 explores these factors and 2.2 take a brief overview of the latest papers.

A. Offloading decision making

Although many factors can affect offloading code, but, the two main criteria affecting this are 1. Performance and 2. Energy consumption. One thing that needs to be clarified here is that the application code can be divided into two distinct parts: one which can never be offloaded to the server (includes user interface and inputs from the environment) and the second one which 'might' be offloaded to the server (does not interact with user or environment and is purely computational in nature).

In this section, we discuss these criteria and how they affect the decision-making process.

1) Performance enhancement

Offloading can be used to enhance response times of complex mobile applications, which require many computations and would take a lot of time if done on a device with very little computation power like mobile phones. A suitable example for this could be a path-finding robot that has to detect obstructions in its path and change its course accordingly. Object and obstruction detection algorithms are usually very complex and require many computations. The processor controlling the robot might not be that fast to run these algorithms and detect these objects in real-time. But if we offload the object detection part to a fast server then it can be done in no time and the robot will avoid colliding with any of those objects. [6]

Another example in context to mobile phones could be of an application which relies heavily on data from different peripherals like GPS, accelerometer, and camera etc. and needs to evaluate the readings from all these collectively. Doing such calculations on the mobile device will be significantly slower than if they are done on a desktop machine. There are also multiple other scenarios in which the performance of mobile devices can be enhanced by using offloading.

Now, we need to define some parameters to establish when offloading code to a server might result in performance enhancement and when it is better to just perform computations on the mobile device. On an abstract scale, we can say that the mobile device's performance will be enhanced if the communication link between the mobile client and server is fast and the amount of data exchanged is smaller in relation to the calculations required. Inequality (1) can be used to describe the relationship between these different parameters: [4]

$$\frac{w}{S_m} > \frac{d_i}{B} + \frac{w}{S_s} \Rightarrow w \times \left(\frac{1}{S_m} - \frac{1}{S_s}\right) > \frac{d_i}{B} \quad (1)$$

Here $\boldsymbol{S_m}$ is the speed of the mobile system, $\boldsymbol{w}$ is the amount of computation that may be offloaded to the server, $\boldsymbol{d_i}$ is the data sent to the server, $\boldsymbol{B}$ is the bandwidth of the channel and $\boldsymbol{S_s}$ is the speed of the server. This inequality holds if we have:

- **large *w*:** The program requires heavy computations
- **large S_s:** The server is fast
- **small d_i:** The data sent to the server is small
- **large B:** The bandwidth is high

Therefore, from the above description it is quite clear that only those parts of code should be offloaded which require heavy computations and very small communication overhead. Otherwise, the performance gain would not be sufficient to make any difference.

ISSN – 1999-4974

2) Energy consumption

Energy is the primary concern for mobile phone users these days. As these phones are not only used for voice communication anymore but their users also use them for acquiring and viewing videos and photographs, playing games, browsing the internet or as personal gadgets etc. All these different uses increase the power consumption of the mobile and reduce battery timings. And even though battery technology has advanced a lot recently, but, it has not been able to keep up with the ever-increasing demand for smaller, lighter and longer lasting batteries. One possible solution here is to offload the more energy consuming operations so that we can save power on our mobile devices [32]. We can use a similar inequality like the previous section to describe the constraints here as well: [4]

$$w \times \left(\frac{P_m}{S_m} - \frac{P_i}{S_s}\right) > P_c \times \frac{d_i}{B} \quad (2)$$

Here $\boldsymbol{P_m}$ is the power on the mobile device, $\boldsymbol{P_c}$ is the power required to transfer data from the mobile to network and $\boldsymbol{P_i}$ is the power consumed at the mobile device while waiting for the results from the server. From analyzing (2), we can see that energy consumption of the mobile device will be minimized when the same requirements as the ones for (1) are met.

However, these inequalities are based on the assumption that the data being transferred is from the mobile device to the server. If the data is already present somewhere on the internet (pictures or videos etc.) and the mobile device only passes the link for that data to the server, then it can fetch that data from the corresponding url and hence increase the performance and reduce the battery consumption as well.

B. Analysis of some leading papers

As mentioned earlier, a lot of research has been done over the past years on the topic of mobile code offloading. Recently with advances in cloud computing and virtualization technology, new doors have been opened in this field as well which have taken computation offloading to a completely new level. In this section, we discuss some of the recent papers, which utilize these technologies to enhance the mobile computing environment. Although there are many papers that discuss this subject but mentioning all of them here would be impossible. Therefore, we picked only the ones that are implementing distinctively different approaches to give the readers a general idea of what the general trends concerning mobile code offloading are these days. Table 1 summarizes the approach used, strengths and weaknesses of each framework.

1) CloneCloud

Its architecture describes a way to partially off-load execution from the smart phone to the computational service infrastructure hosting a cloud of smart phone clones. Smart phone clone at the cloud is a VM of the smart phone OS synchronized with the state of the corresponding smart phone*[33]*. Computationally intensive and background tasks which are less user interactive can be off-loaded to execute on the clone running at a resource rich machine in the cloud. These tasks can be file scanning, photo analysis and web crawling etc. Off-loaded tasks can continue execution even when the smart phone is turned off which greatly helps saving power of the smart phone. CloneCloud uses semi-dynamic partitioning of the code and synchronizes phone with the clone through either fine grain or coarse grain synchronization depending on the off-loaded application requirement and available bandwidth. Updates for synchronization are sent to the clone in the form of deltas to save bandwidth and power. For a practical demonstration, an Android OS application was offloaded to the server where a Dalvik VM [42] was running with the same application. The Replicator running at the smart phone by sending updates to the clone synchronizes clone and smart phone application status. For a practicalapplication, Alien Dalvik [40] can be used to run Android Application on non-Android hardware such as x86 architecture.

2) Cloudlets

Another design is to use cloudlets for code off-loading. Cloudlets are widely spread internet infrastructure whose compute cycles and memory are leveraged by nearby mobile devices. These cloudlets are usually not much resourceful machines but are resource rich compare to the smart phones. Cloudlets can be desktops, net books, kiosks or customized ATM. Cloudlet approach is different from Cloud based approach where smart phones connect to the main cloud, which can be at a multi hop distance. Cloudlets form a peer-to-peer network among themselves along with connecting to the main cloud at the same time*[40]*. Every device connected to the cloudlet is registered at the main cloud and can connect to the main cloud or the cloudlet depending on the throughput and latency. Study [40] shows that for maximum 4 wireless hops from smart phone to the cloudlet, the cloudlet based approach performs poorly for some of the requests, though the cloudlet based approach can outperform the cloud

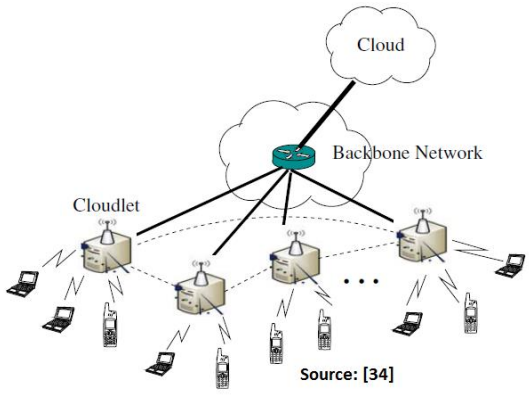

Fig. 1. Cloudlet based Architecture

based approach for most of the requests. The cloudlet-based approach always outperforms the cloud-based approach when there are a maximum of 2 cloudlet hops. In scenarios where the maximum number of cloudlet hops is more than 2 the cloudlet-based approach doesn't always outperform the cloud-based approach. So it is suggested that the cloudlet based approach is to be used when the maximum number of

ISSN – 1999-4974

cloudlet hops does not exceed 2 which can be achieved by using latest technologies such as Flashing [41] or by using Wi-Fi repeaters. Additionally, cloudlets can benefit by keeping routing tables with themselves so the devices of one cloudlet can connect to the devices of other cloudlets or main cloud.

3) MOMCC

Mobile devices are inherently resource poor both in terms of energy and computation power. This paper addresses the later of these two issues by proposing a market based architecture in which nearby mobile devices are used to augment the computation power deficiency of these devices. The basic motivation behind this idea is that most other techniques being proposed require the use of specialized hardware (small servers or high-speed internet connectivity). However, by using this we can eliminate the costly hardware and use the neighboring mobile devices for our computations, as they would have much smaller latencies with our client as opposed to the long latencies experienced in WAN and mobile networks. Although this approach may result in draining the batteries of neighboring mobiles, but as compensation to that, service providers based on how much computations they have performed can pay the owners of the mobile devices, which are performing the computations. In addition, the mobile user who requested the computations will have to pay according to the number of computations offloaded.*[35]*

The basic architecture this paper proposes is one based on and very similar to web services. Every developer that wishes their application to be able to be offloaded should develop it like a web service with a map-reduce like architecture. The overall architecture of the system consists of four distinct entities namely: service developer, service governor, service host and service requester. Service developer is the programmer who develops the application, service host represents the mobile devices, which offer their services for computations, service requestor is the client or the service user and service governor is a central entity, which keeps track of the services, hosts, requestors and distributes the workload between the service hosts.

This kind of publicized computation may result in malicious attacks on users. To stop that a certain level of security needs to be implemented, this is also the job of the governor. The service developer develops an application, registers itself with the governor, and publishes the application there. The service requestors download the application from the governor and the service hosts are published the code they have to execute from the governor. This approach removes any interaction from the developer during execution by totally isolating it from the users, hence, removing any possibility of malicious applications acting as Trojans/spyware etc. The second biggest security risk in this environment is the service hosts. Each of these mobile devices will have different levels of security and reliability on it. For this purpose, the governor constantly monitors these hosts and when assigning a job, only assigns it to hosts, which fulfill the minimum-security criteria for that application.

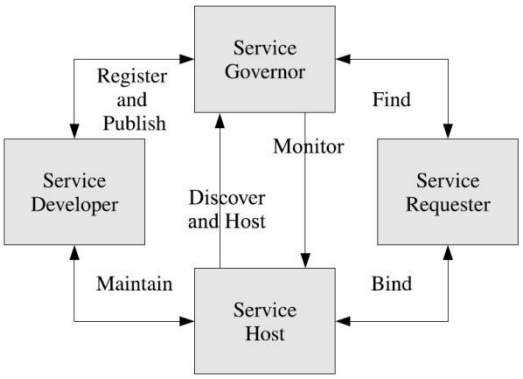

Fig. 2 MOMCC Cycle [35]

This approach can be very attractive and useful for both mobile users and service providers as they can act as service governors while paying service developers and hosts for their services; they can charge the service requestors for the services they request.

4) MAUI

MAUI is a system that enables fine-grained energy-aware offload of mobile code to the infrastructure. Previous approaches to these problems either relied heavily on programmer support to partition an application, or they were coarse-grained requiring full process (or full VM) migration. MAUI uses the benefits of a managed code environment to offer the best of both worlds: it supports fine-grained code offload to maximize energy savings with minimal burden on the programmer. MAUI decides at runtime which methods should be remotely executed. Driven by an optimization engine that achieves the best energy savings possible under the mobile device's current connectivity constraints.*[32]*

MAUI achieves its superior results by some of the benefits of today's latest managed code environments. The authors have used the Microsoft .NET Common Language Runtime (CLR) for their implementation, however, the same could be done through java also. The managed code environment enables it to ignore the instruction set architecture differences between the mobile (ARM) and the offload server (usually x86). First, the CLR is used to generate two copies of the code, one that runs on the client and the other one that runs on the server. Then it uses *program reflection* combined with *type safety* to identify which portions of the code can be offloaded to the server. It also profiles each method to determine its net shipping cost with context to local resources and network conditions and after comparing them only offloads those methods whose offloading can be beneficial in terms of energy conservation and faster execution. All this is performed by the MAUI profiler, which is constantly running in the background. If after some time offloading a method becomes too costly, MAUI can always execute it locally and in the process saving valuable resources.

MAUI provides an architecture in which programmers identify the methods, which can be offloaded to the server, but it is not necessary that they would always be offloaded. Deciding that is the job of the MAUI framework

ISSN – 1999-4974

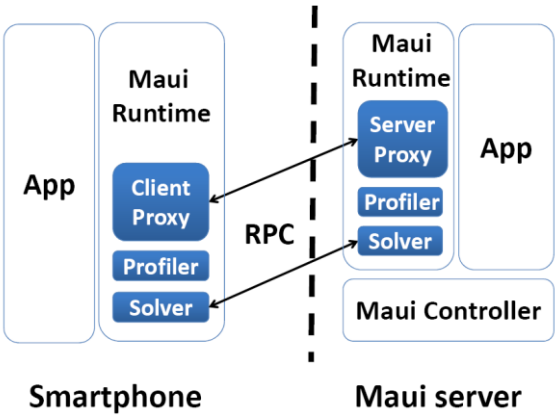

Fig. 3 MAUI Architecture [32]

Figure 3 provides a high-level view of the MAUI system architecture. The MAUI runtime is always running in the background monitoring the program execution. The profiler instruments the program and collects measurements of the program's energy and data transfer requirements. Offload decisions depend on three factors:

- The smart phone device's energy consumption characteristics;
- The program characteristics, such as the running time and resource needs of individual methods
- The network characteristics of the wireless environment, such as the bandwidth, latency, and packet loss.

The MAUI profiler measures the device characteristics at initialization time, and it continuously monitors the program and network characteristics because these can often change and a stale measurement may force MAUI to make the wrong decision on whether a method should be offloaded.

The MAUI solver uses data collected by the MAUI profiler as input to a global optimization problem that determines which remotable methods should execute locally and which should execute remotely. The solver's goal is to find a program partitioning strategy that minimizes the smart phone's energy consumption, subject to latency constraints.

The client and server side proxies handle the data and state transfer between the client and server. Additionally, the MAUI controller present at the server handles authentication and resource allocation for incoming requests.

In addition to the above-mentioned architecture, MAUI also utilizes some optimized programming techniques to minimize the overhead of data transfers between the server and client. For example, during execution at server, instead of sending the whole data to the server every single time, it only sends the difference from previous values (called *deltas*). The results from the server are also sent back in the same format. This approach reduces the amount of communication required resulting in additional energy saving.

Due to the implementation of these techniques, MAUI shows extraordinary results practically. The following figures (figure 4 and 5[32]) display some of the results in terms of energy and execution times.

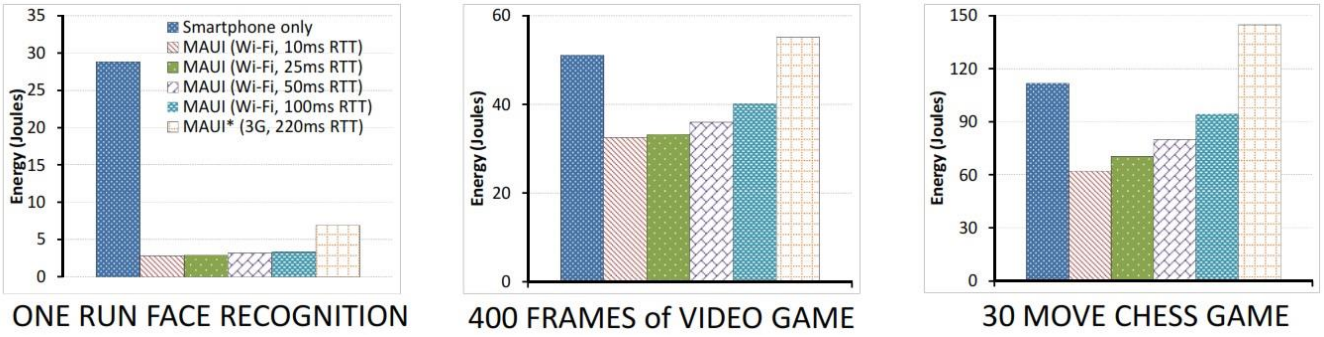

Fig. 4 Energy consumption comparison [32]

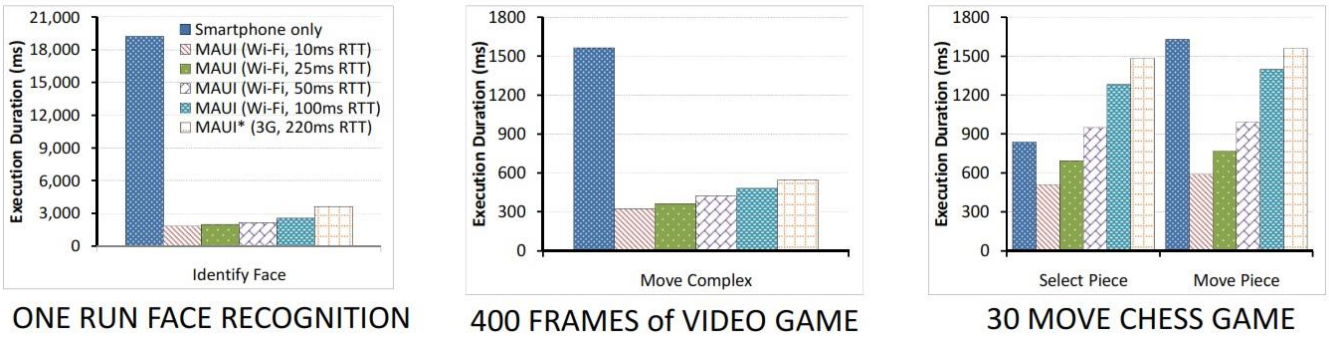

Fig. 5 Execution time comparison [32]

ISSN – 1999-4974

TABLE 1. A Comparison of Different Frameworks

Paper	Decision Making	Parameters Considered	Strengths	Weaknesses
CloneCloud [33]	Semi-dynamic code partitioning	Computation requirement, available bandwidth	• Offloaded tasks keep executing even if phone is turned off. • Updated sent in the form of deltas. • Resource rich server.	• Mobile device's clone is present at the cloud server, large hop distances and connectivity issues can introduce delays between updates.
Cloudlets [40]	Dynamically, depending on network conditions.	Throughput and Latency	• Peer to peer network reduces latency. • Outperforms cloud based approach for up-to 2 hops. • Can fall back to use the cloud server if no cloudlets available nearby.	• Cloudlets usually less resource rich than cloud servers. • Moving virtual machines between cloudlets is an extra overhead. •
MOMCC [35]	Developer specified code partitioning, governed by service provider.	Available hosts fulfilling the minimum requirements.	• Easier to develop. • Lower latencies.	• Drains batteries of host devices. • Multiple possible security issues. • Requires constant monitoring from the service providers.
MAUI [32]	Dynamically through MAUI profiler	The device's energy consumption characteristics. Running time and required resources of individual methods. Network characteristics.	• Updates sent in the form of deltas. • VM running on cloud in parallel to the mobile device. • Only offloads if it results in energy conservation and faster execution. • Performs a lot better than previous approaches.	• Performance drops at higher latencies. • High overhead due to continuous profiling

III. ENABLING TECHNOLOGIES

This section describes some of the enabling technologies for the mobile computation offloading environments. The introduction of these technologies has made it possible for code offloading to be realized by offering improvements in both architecture and infrastructure. The major factors affecting this are advancements in wireless network architectures, cloud computing and virtualization. Here we describe these briefly and analyze how these have helped improve code offloading. However, these two are not the only factors and only represent the authors' point of view.

A. Wireless networks and mobile agents

Until the late 90s, mobile networks did not have much speed and the communication was full of errors and had heavy losses. However, with the introduction of new technologies (3G, Wi-Fi etc.), the problem with speed is pretty much solved and with the introduction of even faster network technologies like 4G, speeds are expected to become even faster. These improvements spurred many research activities on mobile computing, including mobile agents.

Mobile agents are autonomous programs that can control their movement from machine to machine in a heterogeneous network. Mobile Agent infrastructures work to remove the platform dependence while working in a mobile environment. They usually make use of platform independent technologies like XML or Java [12,13,18,19]. All these technologies focus on migrating computation for mobile devices, network connectivity, and developing platform independent applications.

B. Virtualization and cloud computing

Virtualization is a very old technology initially introduced by IBM as a means to manage mainframe computers and their usage [38] but was soon forgotten due to the introduction of cheaper and smaller x86 machines [39]. However, these x86 machines also come with problems like underutilization, operational costs and security risks. During the last decade, virtualization has re-emerged as a solution to all these problems. Virtualization provides solutions to all these problems by running multiple operating systems on a single machine simultaneously, which are concurrent but totally isolated from each other. Many different kinds of virtual machines can be created on a single machine making it highly scalable.

Cloud computing takes the concept of virtualization to a whole new level by providing users with instances of virtual machines on '*lease*' whose number can be increased or decreased according to the user's requirements. These cloud-computing environments can be used very effectively for the purpose of code offloading due to the services and ease of use they provide for the developers and how they are already optimized for dynamic changes in network and bandwidth utilization.

IV. DISCUSSION

The major aim of this paper is to present the major research that has been done in the field of code offloading and to evaluate if code offloading (in its current form) feasible in the industry. Keeping table 1 in mind, following are some of the major areas of concern we have identified.

ISSN – 1999-4974

A. Lack of Infrastructure

The biggest obstacle in adapting code offloading is the limited infrastructure present in the industry today. Code offloading is viable only in conditions where the server (the processing unit to which the code is offloaded) is very near the client. As the number of hops between the client and the server increase, the efficiency of code offloading decreases.

In addition, in the case of VM based code offloading it is assumed that the server would have the necessary software/hardware specifications to successfully run the code in the VM. To achieve in the industry (on a wide scale) is very difficult. The primary reason is the sheer number of VM's that are needed: iOS, Android, Windows Mobile and the sheer versions of each platform.

B. High Speed Connectivity

As we have demonstrated above, the power consumption is inversely proportional to the available bandwidth. Bandwidth available over data networks (3G, Edge, and GPRS) is not sufficient for optimal code offloading as the energy conserved by offloading code is offset by energy consumed by data transfer.

To make code-offloading energy efficient we would need data networks, which are very fast (near the speeds of Wi-Fi).

C. Lack of Development Technologies

In code offloading a lot of code is being executed in parallel on both the server and the client, also at the end of each execution cycle the states/values of both client and server need to be synchronized together. This presents another challenge: the lack of development tools to help in the development and debugging such parallel executions.

By default, all programmers program their code to run sequentially. Even though parallel processing is common these days, most programming languages are still sequential. The major reason is the difficulty in debugging

The same is the problem is with code offloading. Today we lack the development tools to develop such applications where code is offloading automatically, run in parallel, and lack the ability to debug it thoroughly.

D. Subnet Switching

A major problem with all the current implementations of code offloading is how the state of the server is transferred from one cell to the next. This is especially true in cases where the server is coupled very closely with the cell in which the client currently is.

As the user moves from one node to the next, its connection to the server is broken. If the server remains at the same node, then the hops between it and the client increase: thus, decreasing the performance and benefits of code offloading. One way to overcome this is that the server moves with the client to next the code. Here the problem is how would the server know to which node it has to shift to, and how would it transfer it state.

V. CONCLUSION AND FUTURE WORK

As we have described above, code offloading in its current state is not ready to be adapted by the industry on a wide scale. However, there have been cases in the recent history where industry has adapted code offloading quite successfully. The industry adapted the traditional client/server model into code offloading quite successfully and some of the examples are Siri (a digital assistant provided by Apple in its flagship product: iPhone), Shazam (a song recognizing software). In both these cases the programmers used the traditional client/server model to offload parts of the program (such as speech recognition) to the server.

In order to be able to successfully adapt code offloading on a wide scale, the following points need to be addressed in further studies:

1. How to switch between nodes more efficiently?
2. Improving the development technologies.
3. How to offload code more reliably and when is the ideal time to offload code?

REFERENCES

[1] R. A. Powers. Batteries for low power electronics.Proceedings of the IEEE, 83:687–693, April 1995.

[2] CNN.com, ―Battery Life Concerns Mobile Users,‖ 23 Sept. 2005.

[3] J. Paczkowski, ―Iphone Owners Would Like to Replace Battery,‖ All Things Digital, 21 Aug. 2009.

[4] Kumar, Karthik et al. "A Survey of Computation Offloading for Mobile Systems." *Mobile networks and Applications* (2012): 1-12.

[5] Hong YJ, Kumar K, Lu YH (2009) Energy efficient content based image retrieval for mobile systems. In: International symposium on circuits and systems, pp 1673–1676

[6] Nimmagadda Y, Kumar K, Lu Y-H, Lee CSG (2010) Realtime moving object recognition and tracking using computation offloading. In: IEEE international conference on intelligent robots and systems, pp 2449–2455

[7] Ou S, Yang K, Liotta A, Hu L (2007) Performance analysis of offloading systems in mobile wireless environments. In: IEEE international conference on communications, pp 1821–1806

[8] Xian C, Lu Y-H, Li Z (2007) Adaptive computation offloading for energy conservation on battery-powered systems. In: International conference on parallel and distributed systems, pp 1–8

[9] Wolski R, Gurun S, Krintz C, Nurmi D (2008) Using bandwidth data to make computation offloading decisions. In: IEEE international symposium on parallel and distributed processing, pp 1–8

[10] Huerta-Canepa G, Lee D (2008) An adaptable application offloading scheme based on application behavior. In: International conference on advanced information networking and applications - workshops, pp 387–392

[11] Forman GH, Zahorjan J (1994) The challenges of mobile computing. Computer 27(4):38–47

[12] Joseph AD, de Lespinasse AF, Tauber JA, Gifford DK, KaashoekMF (1995) Rover: a toolkit for mobile information access. In: ACM symposium on operating systems principles, pp 156–171

 ISSN – 1999-4974

[13] Kotz D, Gray R, Nog S, Rus D, Chawla S, Cybenko G (1997) Agent Tcl: targeting the needs of mobile computers. IEEE Internet Comput 1(4):58–67
[14] Noble BD, Satyanarayanan M (1999) Experience with adaptive mobile applications in Odyssey. Mobile NetwAppl 4(4):245–254
[15] Perkins CE (1996) Handling multimedia data for mobile computers. In: Computer software and applications conference,pp 147–148
[16] Qi M (1997) Resource conservation in a mobile transaction system. IEEE T Comput 46:3:299–311
[17] White JE (1997) Mobile agents. In: Software agents (MIT Press), pp 437–472
[18] Wong D, Paciorek N, Moore D (1999) Java-based mobile agents. Commun ACM 42(3):92–102
[19] Wong D, Paciorek N, Walsh T, DiCelie J, Young M, Peet B (1997) Concordia: an infrastructure for collaborating mobile agents. In: International workshop on mobile agents, pp 86–97
[20] O'Hara KJ, Nathuji R, Raj H, Schwan K, Balch T (2006) Autopower: toward energy-aware software systems for distributed mobile robots. In: IEEE international conference on robotics and automation, pp 2757–2762
[21] Wang C, Li Z (2004) Parametric analysis for adaptive computation offloading. In: ACM SIGPLAN conference on programming language design and implementation, pp 119–130
[22] Rong P, Pedram M (2003) Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In: Conference on design automation, pp 906–911
[23] Li Z, Wang C, Xu R (2002) Task allocation for distributed multimedia processing on wirelessly networked handheld devices. In: Parallel and distributed processing symposium, pp 79–84
[24] Li Z, Wang C, Xu R (2001) Computation offloading to save energy on handheld devices: a partition scheme. In: International conference on compilers, architecture, and synthesis for embedded systems, pp 238–246
[25] Gu X, Nahrstedt K, Messer A, Greenberg I, Milojicic D (2003) Adaptive offloading inference for delivering applications in pervasive computing environments. In: IEEE international conference on pervasive computing and communications, pp 107–114
[26] Goyal S, Carter J (2004) A lightweight secure cyber foraging infrastructure for resource-constrained devices. In: Mobile computing systems and applications, pp 184–195
[27] Ou S, Yang K, Hu L (2007) Cross: a combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments. In: IEEE global telecommunications conference, pp 720–725
[28] Rim H, Kim S, Kim Y, Han H (2006) Transparent method offloading for slim execution. In: International symposium on wireless pervasive computing, pp 1–6
[29] Seshasayee B, Nathuji R, Schwan K (2007) Energy aware mobile service overlays: cooperative dynamic power management in distributive systems. In: International conference on automatic computing, pp 6–12
[30] Weinsberg Y, Dolev D, Wyckoff P, Anker T (2007) Accelerating distributed computing applications using a network offloading framework. In: IEEE international parallel and distributed processing symposium, pp 1–10
[31] Yang K, Ou S, Chen H-H (2008) On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. IEEE communications magazine 46(1):56–63
[32] Cuervo, Eduardo et al. "MAUI: making smartphones last longer with code offload." *Proceedings of the 8th international conference on Mobile systems, applications, and services* 15 Jun. 2010: 49-62.
[33] Chun, Byung-Gon et al. "Clonecloud: elastic execution between mobile device and cloud." *Proceedings of the sixth conference on Computer systems* 10 Apr. 2011: 301-314.
[34] Satyanarayanan, Mahadev et al. "The case for vm-based cloudlets in mobile computing." *Pervasive Computing, IEEE* 8.4 (2009): 14-23.
[35] Abolfazli, Saeid et al. "MOMCC: Market-Oriented Architecture for Mobile~ Cloud~ Computing Based on Service~ Oriented~ Architecture." *arXiv preprint arXiv:1206.6209* (2012).
[36] Cidon, Asaf et al. "MARS: adaptive remote execution for multi-threaded mobile devices." *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds* 23 Oct. 2011: 1.
[37] Kumar K, Lu YH (2010) Cloud computing for mobile users: can offloading computation save energy? IEEE Comput 43(4):51–56
[38] Goldberg RP (1974) Survey of virtual machine research.IEEEComput 7(6):34–45
[39] Rosenblum M, Garfinkel T (2005) Virtual machine monitors: current technology and future trends. IEEE Comput 38(5):39–47
[40] DebessayFesehaye, Yunlong Gao, Klara Nahrstedt Impact of Cloudlets on Interactive Mobile Cloud Applications

 ISSN – 1999-4974